International Journal of Advancements in Electronics and Electrical Engineering– IJAEEE
Volume 4 : Issue 1 [ISSN : 2319-7498]
Publication Date : 30 April, 2015

Pong Game as an Embedded System

Sadiye Nergis Tural-Polat

Abstract— **In this study, a Pong game running on an FPGA development and education board using VHDL hardware description language is synthesized. To this end, an original pong game code that works solely on the FPGA board is designed. One important aspect of this design is that the code does not use any SRAM modules for storage. Though Altera DE0 development and education board is used for the implementation, thanks to the usage of only the standard VHDL functions, it can run on any other FPGA boards.**

Keywords— **pong game, game code in VHDL, embedded systems.**

I. Introduction

It is well known that FPGA technology provides a suitable framework for designing and implementing complex systems in a relatively short time. It is also a very good medium for real-time image processing applications.

Pong is one of the earliest arcade video games; it is a tennis sports game featuring simple two-dimensional graphics. Pong was one of the first video games to reach mainstream popularity. The aim is to defeat an opponent in a simulated table-tennis game by earning a higher score. The game was originally manufactured by Atari Incorporated (Atari), who released it in 1972. Allan Alcorn created Pong as a training exercise assigned to him [1].

Pong quickly became a success and is the first commercially successful arcade video game machine, which helped to establish the video game industry along with the first home console, the Magnavox Odyssey. The company released several sequels that built upon the original's gameplay by adding new features. The game has been remade on numerous home and portable platforms following its release [2].

In this paper a fully-embedded Pong game code that runs on an FPGA development board is designed. All parts of the game code are written in VHDL and synthesized for the dedicated FPGA chip on the DE0 board using Altera Quartus II software. The Modelsim program is used for the simulations.

The paper is organized as follows: The second chapter outlines the main Pong game functions. The VHDL code that achieves the desired functions is given in chapter III and the results are summarized with the conclusion in chapter IV.

Sadiye Nergis Tural-Polat
Yildiz Technical University
Turkey

II. The Game of Pong

Pong is a two-dimensional sports game that simulates table tennis. The player controls an in-game paddle by moving it vertically across the left side of the screen, and can compete against either a computer-controlled opponent or another player controlling a second paddle on the opposing side. Players use the paddles to hit a ball back and forth. The aim is for each player to reach eleven points before the opponent; points are earned when one fails to return the ball to the other (Fig. 1) [1][2].

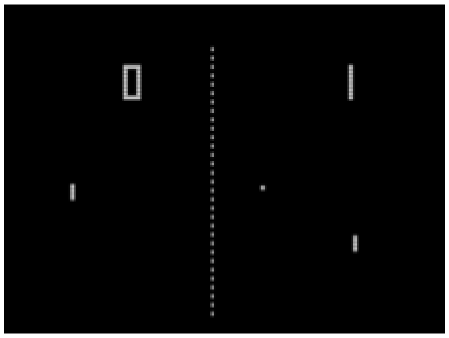

Figure 1. The original Pong game display. The players move the paddles up and down to hit the ball, the score is kept by the numbers (0 and 1) at the top of the screen [1].

III. Pong Game Code in VHDL

We synthesized a one player version of the pong game code in VHDL. In our version, the player moves a paddle left and right near the bottom of the screen to hit a moving ball and the ball bounces back from the left and right edges of the screen. There are a total of 16 bricks arranged at equidistance from one another near the top of the screen. The objective of the game is to hit the bricks with the ball without dropping the ball to the bottom edge. The brick hit by the ball disappears from the screen. The game is lost if the ball falls at the bottom edge of the screen (Fig. 2).

The designed VHDL code that realizes the desired pong game functions consists of three main parts. The first part generates the necessary clock signals for the game from the on-board 50 MHz clock signal of DE0 Board [3]. The second part constructs the VGA image compatible to VGA Standards for the game display and the third part produces and controls the main Pong Game Functions (Fig. 3).

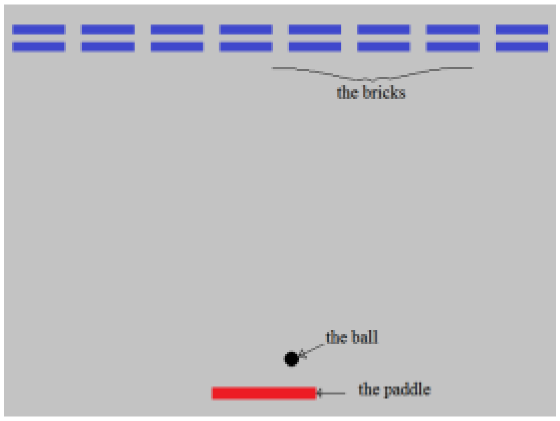

Figure 2. The main pong display screen

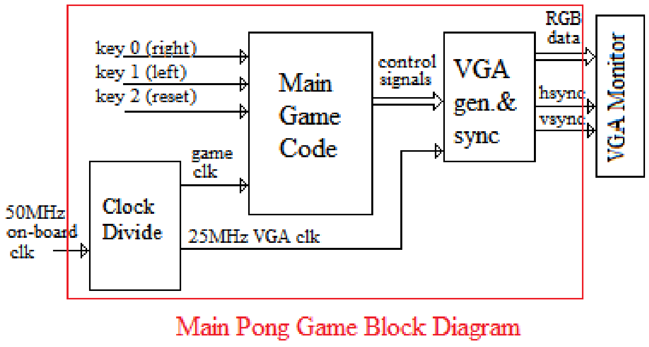

Figure 3. Main block diagram of the VHDL Pong Code

A. *System Clock and Game Code Clock*

As stated above, the DE0 board has 50 MHz on-board clock. To be able to construct VGA image, we need to have a 25MHz VGA synchronization clock. What is more, to be able to obtain animated image sequences for game play, it is necessary to move the objects in the frame at an observable speed to human eye. We choose the game clock to be 60Hz for this design.

B. *VGA Image Construction and Synchronization*

VGA (Video Graphics Array) video consists of sequential image frames moving in time. Each frame consists of vertically arranged rows of horizontal pixels. Every image frame is constructed by scanning the pixels from the top left row to the right until the end of line, then repeating the process for the next line until the end of the frame [4].

Standard VGA image has 640x480 pixel resolution and 60Hz refresh rate. These pixels are in the visible range, several more pixels are used for synchronization purposes; therefore the actual pixel size is 800x525. Hence the 25MHz pixel clock is obtained as

$$f_{VGA} = 800x525x60 \cong 25MHz \qquad (1)$$

Figure 3 shows horizontal synchronization timing specification and number of pixels used for horizontal and vertical synchronization are given in Table 1. The proper synchronization signals are obtained by using two counters for horizontal and vertical pixel positions.

DE0 Board has 4-bit DAC for its VGA output RGB (Red, Green, and Blue) signals, therefore R, G and B signals are constructed as 4-bit signals in the VHDL code.

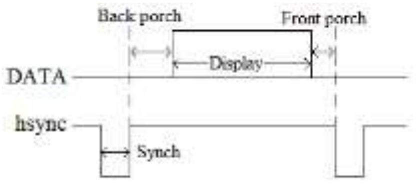

Figure 4. Horizontal timing specification of VGA image [7]

TABLE I. HORIZONTAL AND VERTICAL SYNCH PIXEL SIZES

Synch Signals	**pixels**	
	Horizontal	***Vertical***
Synch	96	2
Back Porch	48	33
Front Porch	16	10
Display	640	480
Total	800	525

There is a pixel generation block in the VGA Sync block. The pixel generation block produce the 3-bit RGB signal which depends on the control signals from the main game code and the pixel x and y coordinates for each pixel in the VGA frame. For 640-by-480 VGA resolution, there are about 310k pixels on a screen. This translates to 930k memory bits for a RGB color display. To reduce the memory requirement, one alternative is to use a tile-mapped scheme. In this scheme, a collection of bits are grouped to form a tile and each tile is treated as a display unit. For the pong game, the video display is very simple and contains only a few objects. Similar to the tile-mapped scheme, instead of wasting memory to store a mostly blank screen, we can generate these objects using simple object generation circuits for the paddle, the ball and the bricks. Then we can determine if the current pixel is within the defined object boundaries and choose the RGB values (ie. the color of the current pixel) accordingly (Fig. 5).

The object generating code keeps the current location of the object (for the still objects such as bricks the current location is the same at all times but for the moving objects such as the ball the location changes between the image frames), compares it with the current pixel location and if the pixel location falls within the object location, the object is selected by object_on signal and the appropriate RGB color is assigned to the pixel through the rgb mux.

The rectangular objects are defined by their boundary coordinates on the screen. Direct object boundary checking for the nonrectangular objects is very difficult [5] and is not preferred in this design. Instead, we use a bitmap to specify the circular ball object and generate the ball_on and ball_rgb signals according to the bitmap stored in an array in the code (Fig. 6).

International Journal of Advancements in Electronics and Electrical Engineering– IJAEEE
Volume 4 : Issue 1 [ISSN : 2319-7498]
Publication Date : 30 April, 2015

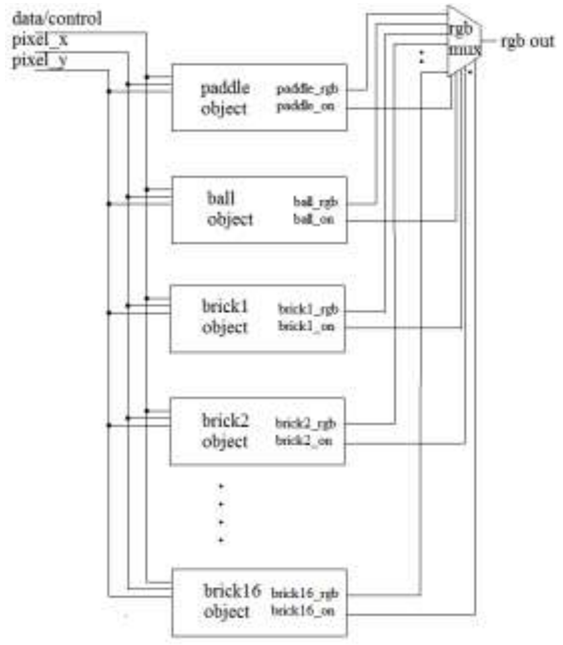

Figure 5. Object-mapped scheme

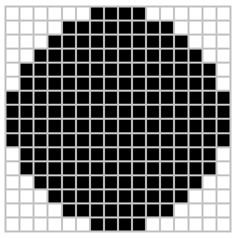

Figure 6. The circular ball object bitmap

C. *Illusion of Animation*

When an object changes its location gradually in each frame, it creates the illusion of motion and becomes animated. To achieve this, we use registers to store the boundaries of an object and update its value in each VGA frame. In the pong game, the paddle is controlled by two pushbuttons and can move to the left and right, and the ball can move and bounce in all directions. If the ball hits one of the boundaries of a brick, the brick should disappear in the next frame and the background color should be displayed in its place.

The move of the paddle is controlled by the key 0 (right) and key 1 (left) of the DE0 board. So a small Finite State Machine checks if the buttons pressed and if there is available space, moves the paddle to the right or left (changing only two boundaries of the paddle since it cannot move up or down) accordingly.

The move of the ball is a bit more complex. The four boundaries of the ball should be changed at each frame to achieve ball animation and also should be checked for collisions to the screen edges, bricks or the paddle.

D. *Main Game Code*

The main Finite State Machine (FSM) of the program consists of twelve states (Reset, Start, Key0, Key1, Bottom_edge, Left_edge, Right_edge, Paddle, Brick, No_change, Win, Loose) and 18 state transitions. State diagram of the main FSM is given in Fig. 7.

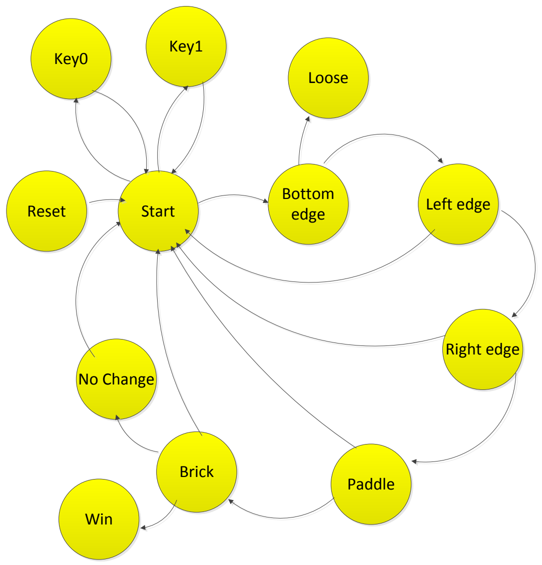

Figure 7. Main FSM State Diagram

Reset state is the initial state, the initial values of all the registers and object boundaries are set in this state. The ball is generated at the top center of the screen just below the lower line of the bricks and starts its downward motion. The paddle is also at the bottom center of the screen. The next state is the Start.

At the Start state the code checks if any of the keys on the DE0 board are pressed by the user and checks the movement of the ball and the state transitions are done accordingly. Key 0 controls the right move of the paddle (so if key 0 is pressed, the next state becomes Key0), key 1 controls the left move of the paddle (Key1 state). If none of the keys are pressed, (or after the appropriate paddle motion) the next state is Bottom_edge. When the appropriate conditions are met (explained below) in every other state or when the reset switch is pressed (Key2), the program returns to the start state.

At the Key0 state the code checks if there is enough space to move the paddle to the right and moves the paddle by changing the boundaries 10 pixels to the right. The next state is the Bottom_edge.

At the Key1 state the code checks if there is enough space to move the paddle to the left and moves the paddle by changing the boundaries 10 pixels to the left. The next state is the Bottom_edge.

At the Bottom_edge state, the code checks if the bottom boundary of the ball touches the bottom edge of the screen which means the game is over. The next state becomes the Loose state. If not, the code checks if the ball reached at the left edge of the screen and the next state becomes the Left_edge.

The Left_edge state checks if the ball reaches at the left edge of the screen. Then the ball should bounce back vertically at the same direction with the symmetrical angle with which it collides to the edge. The new boundaries of the ball are set accordingly. Then the code checks if the ball hit the right edge of the screen, therefore the next state becomes the Right_edge.

Similarly, the Right_edge state examines whether the ball reaches at the right edge of the screen. Then the ball should bounce back vertically at the same direction with the symmetrical angle with which it collides to the edge. The new boundaries of the ball are set accordingly. The next state is the Paddle.

The Paddle state checks if the ball touches to the paddle. Similar to the edge states, the ball should bounce back (horizontally) at the same direction with the symmetrical angle with which it collides to the paddle. The new boundaries of the ball are set accordingly. Then the code checks if the ball hit one of the bricks so the next state is the Brick.

The Brick state inspects whether the ball touches to the one of the bricks. Now, both the brick should disappear and the ball should bounce back horizontally at the symmetrical angle with which it collided to the brick. The state of the bricks (present or not) are stored in an array in the code so the array is updated accordingly and the new boundaries of the ball is set. If all the bricks are hit without dropping the ball to the bottom edge, the game is won therefore the next state is the Win state, otherwise the next state is the No_change.

If the ball does come into contact with none of edges or the objects, it should continue its movement as it is. This is done in the No_change state and the code returns to the Start state.

In the Win state the game is stopped and "WIN" text is written on the screen by using bitmaps for the three letters stored in arrays in the code. Hitting the reset button (Key2) restarts the game.

In the Loose state the game is stopped and "LOOSE" text is written on the screen by using bitmaps for the letters stored in arrays in the code. Hitting the reset button (Key2) restarts the game.

The game code flowchart is given in Fig. 8.

The designed pong game code is approximately 1100 lines long, the compilation of the code takes approximately 21 minutes and the code uses about 70% of the logic elements of the FPGA.

IV. Conclusion

In this study, we developed an original pong game code that runs entirely on an FPGA chip in VHDL. It can run on its own without requiring any computer interruption once programmed, thus provides mobility. The code does not use any SRAM modules for storage and that improves the speed of the code. The implementation is done on the Altera DE0 board but since the code is written using standard VHDL functions, it can run on any other FPGA boards. Improvements can be added such as score evaluation and inclusion of several ball speed levels according to the number of hits on the edges.

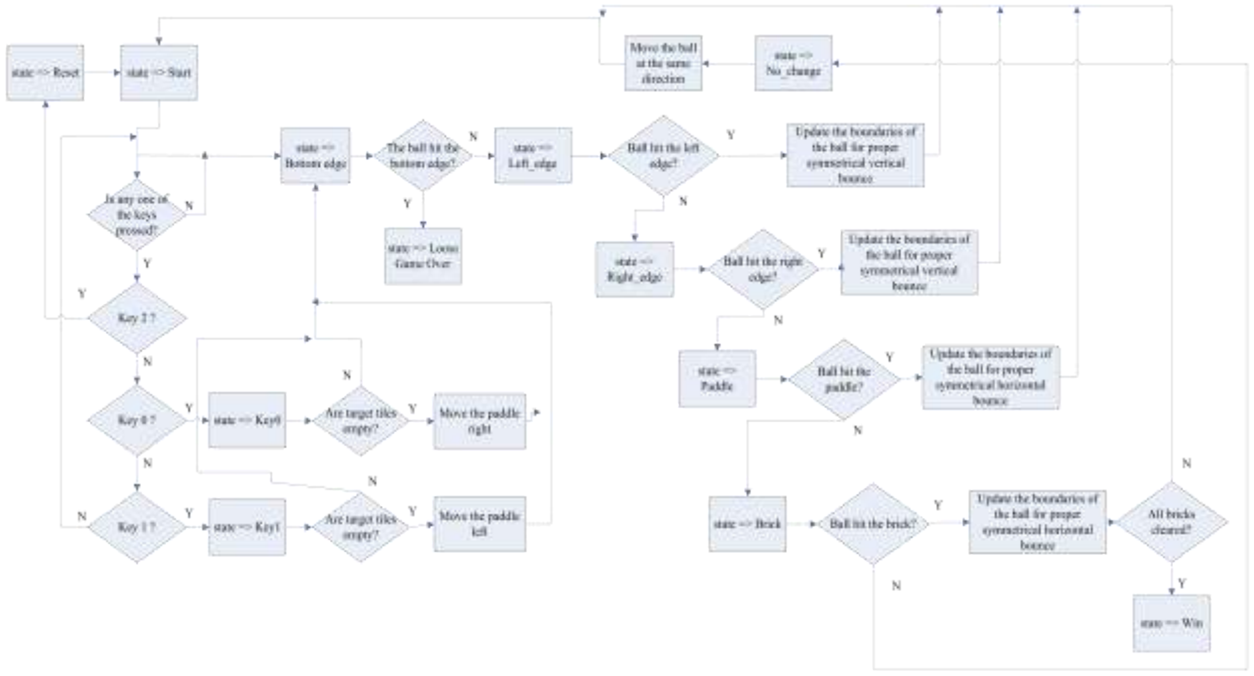

Figure 8. Tetris Code Flowchart

References

[1] Sellers, John (August 2001). "Pong". Arcade Fever: The Fan's Guide to The Golden Age of Video Games. Running Press. pp. 16–17. ISBN 0-7624-0937-1.

[2] Kent, Steven (2001). "And Then There Was Pong". Ultimate History of Video Games. Three Rivers Press. pp. 40–43. ISBN 0-7615-3643-4.

[3] Terasic Technologies, Altera DE0 Development and Education Board User Manual, 2009.

[4] Pong P. Chu, FPGA Prototyping by VHDL examples, New-Jersey, USA: John Wiley & Sons, 2008.

[5] D. L. Perry, VHDL Programming by Example, 4th ed., New York, USA, McGraw-Hill, 2002.